

# A continuing overview of STAT 20 R

JE Hug

9/16/2020

Welcome to this document created by Josh Hug, one of the GSI's for STAT 20. This document will try and keep up with our R progress as we go on and I will add in depth (time permitting) explanations of what we are doing as I go on. Remember to check github for the latest updates to this document as it progresses!

A generic glossary of R commands that we have used so far.

```
# these are the packages we are using so far
```

```
library(dplyr)
library(ggplot2)
```

I'll begin with some simple use of the main dplyr functions we use, on the palmer penguins data set (make sure to install it if you don't have it yet). I prefer using this dataset over something like iris due to the fact that while iris is a classic dataset, it was compiled by Ronald Fisher (a prominent eugenicist) and published in a eugenics journal originally. This dataset provides a nice alternative with similar properites.

## 1 Filter, Select, Mutate Basics

The key facts here are to use select if we want

```
# Here I will use the palmer penguins data set
```

```
# if you don't have it installed
```

```
# install.packages("palmerpenguins")
```

```
library(palmerpenguins) # where this data set comes from
```

```
glimpse(penguins) # a nice function to take an easy look at the data
```

```
## Rows: 344
## Columns: 8
## $ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A...
## $ island       <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge...
## $ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34....
## $ bill_depth_mm <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18....
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, ...
## $ body_mass_g   <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 347...
## $ sex          <fct> male, female, female, NA, female, male, female, m...
## $ year         <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2...
```

Suppose I wanted to make a new column bill\_length\_cm and body\_mass\_kg (where I convert units into cm and kg respectively)

We can use mutate to add a new column as some function of another column

```
new_pen <- mutate(penguins, bill_length_cm = bill_length_mm / 10, body_mass_kg = body_mass_g / 1000 )  
  
glimpse(new_pen)
```

```
## Rows: 344  
## Columns: 10  
## $ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A...  
## $ island       <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge...  
## $ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34....  
## $ bill_depth_mm <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18....  
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, ...  
## $ body_mass_g    <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 347...  
## $ sex           <fct> male, female, female, NA, female, male, female, m...  
## $ year          <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2...  
## $ bill_length_cm <dbl> 3.91, 3.95, 4.03, NA, 3.67, 3.93, 3.89, 3.92, 3.4...  
## $ body_mass_kg   <dbl> 3.750, 3.800, 3.250, NA, 3.450, 3.650, 3.625, 4.6...
```

Now suppose I want penguins that weigh less than like 3000 g (3kg) only. Since this is subsetting over rows with a specific condition we use the filter function from dplyr.

```
new_pen_light<- filter(new_pen, body_mass_kg <3 )  
  
glimpse(new_pen_light)
```

```
## Rows: 9  
## Columns: 10  
## $ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A...  
## $ island       <fct> Dream, Biscoe, Biscoe, Biscoe, Dream, Biscoe, Tor...  
## $ bill_length_mm <dbl> 37.5, 34.5, 36.5, 36.4, 33.1, 37.9, 38.6, 43.2, 46.9  
## $ bill_depth_mm <dbl> 18.9, 18.1, 16.6, 17.1, 16.1, 18.6, 17.0, 16.6, 16.6  
## $ flipper_length_mm <int> 179, 187, 181, 184, 178, 193, 188, 187, 192  
## $ body_mass_g    <int> 2975, 2900, 2850, 2850, 2900, 2925, 2900, 2900, 2700  
## $ sex           <fct> NA, female, female, female, female, female, femal...  
## $ year          <int> 2007, 2008, 2008, 2008, 2008, 2009, 2009, 2007, 2008  
## $ bill_length_cm <dbl> 3.75, 3.45, 3.65, 3.64, 3.31, 3.79, 3.86, 4.32, 4.69  
## $ body_mass_kg   <dbl> 2.975, 2.900, 2.850, 2.850, 2.900, 2.925, 2.900, ...
```

## 2 Histograms in R

Take a look at this cheat sheet, you're probably extremely overwhelmed by this and personally I don't know what at least half of the stuff on this page does but it will save you a lot of time from googling. There are actually cheat sheets for most tidyverse (what these packages are a part of) packages so you can check out ones for dplyr and such.

### 2.1 General format of ggplot

The general format of ggplot is that we call some generic function, then we can just build on it by (literally) adding to it other components. Now we always start with the base ggplot function which has two main inputs the dataframe and then the aesthetic. ggplot is extremely flexible and I can't begin to scratch the surface of what you can do here so take a look at some other examples or the cheat sheet.

Suppose I want a histogram of penguins bill length in mm. I tell ggplot to look at my dataframe penguins and then I tell it to look for a specific column by its column name. Note that I don't have to subset my

dataframe at all before this if I want to use entire columns, no matter how many other columns ggplot only looks at the ones I tell it to.

```
ggplot(penguins, aes(x=bill_length_mm))
```

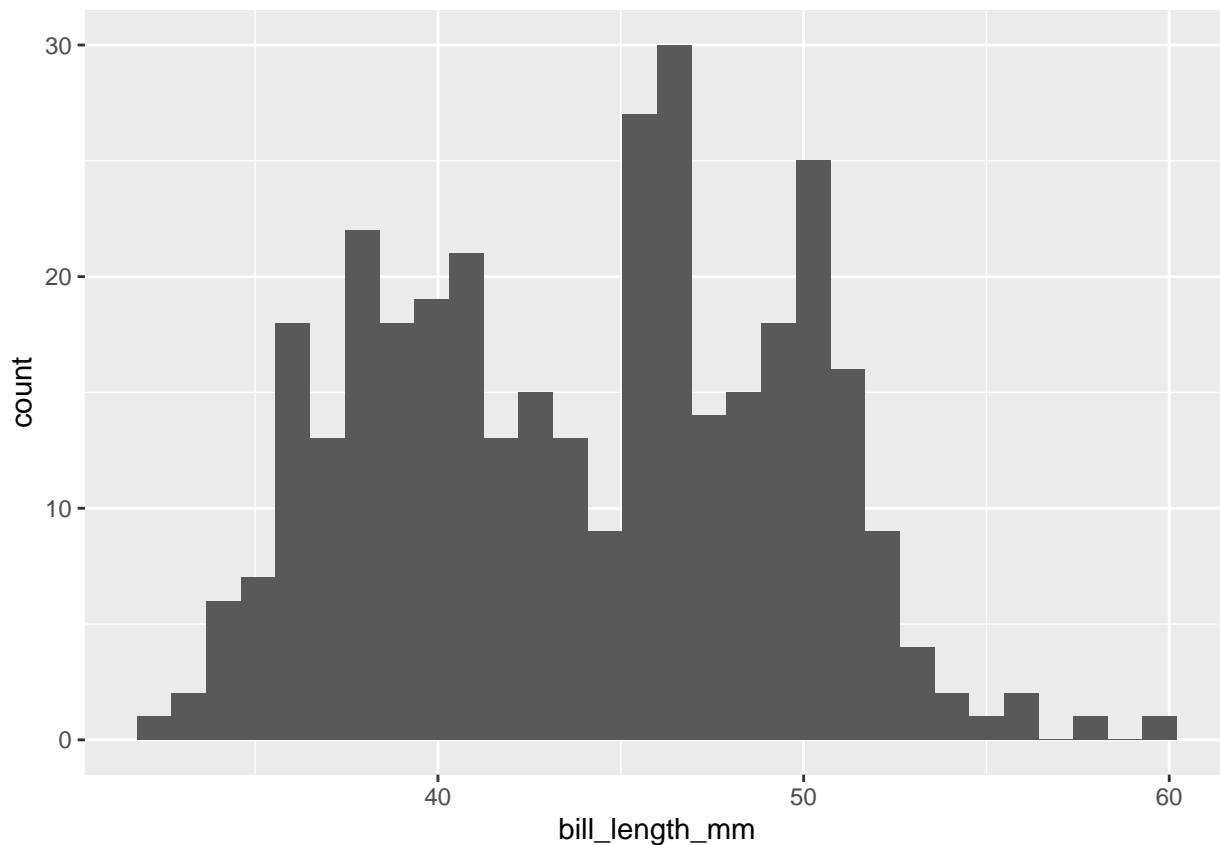


So R here has outputted nothing because I have told it only what the data frame and the column I want inputted is but I haven't specified which type of plot I want. So my next line I literally add onto it to tell R to make a histogram

```
ggplot(penguins, aes(x=bill_length_mm)) +  
  geom_histogram() # this is the line that tells it to make a histogram
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```

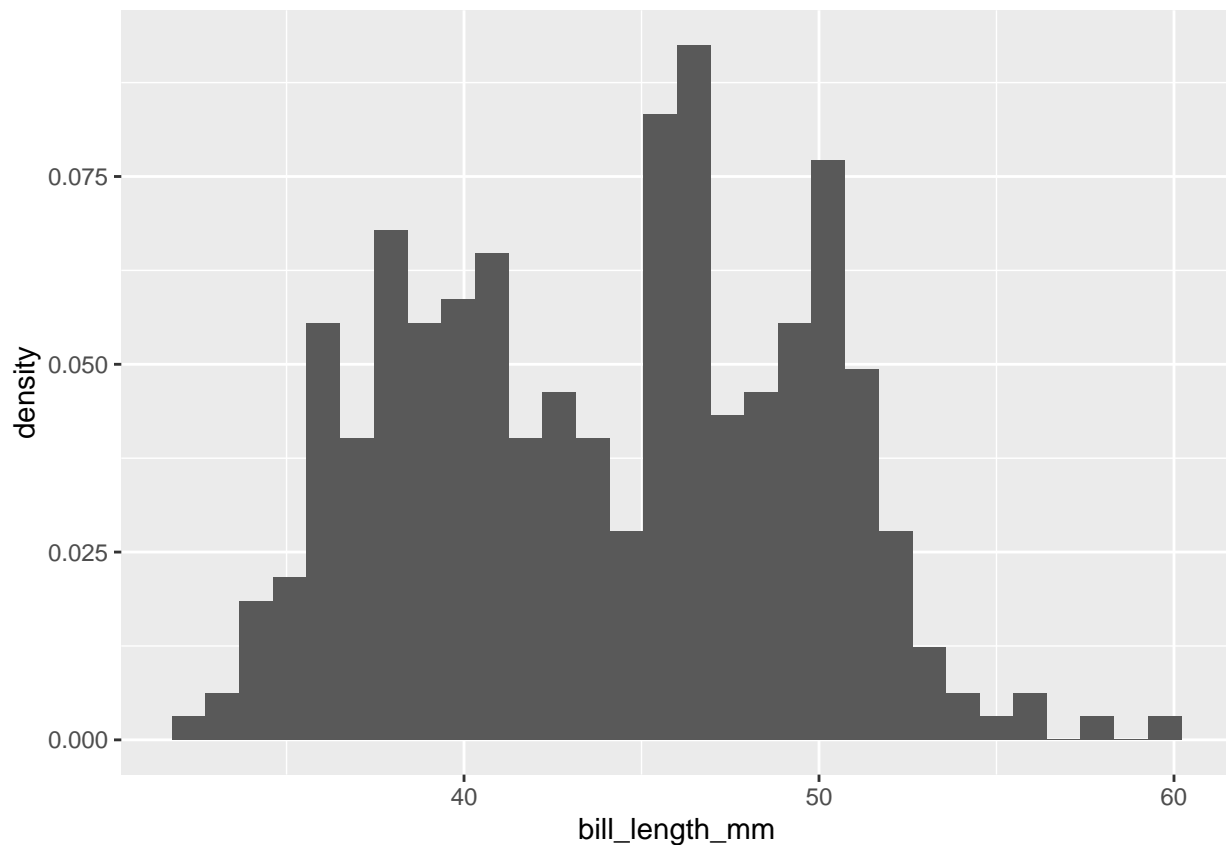


So R has made a histogram here but notice that we don't have density on the y axis we just have counts which is not what we usually want, so we can add another argument to aes that specifically tells it to use the density on the y axis.

```
ggplot(penguins, aes(x=bill_length_mm, y=..density..)) +  
  geom_histogram() # this is the line that tells it to make a histogram
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```

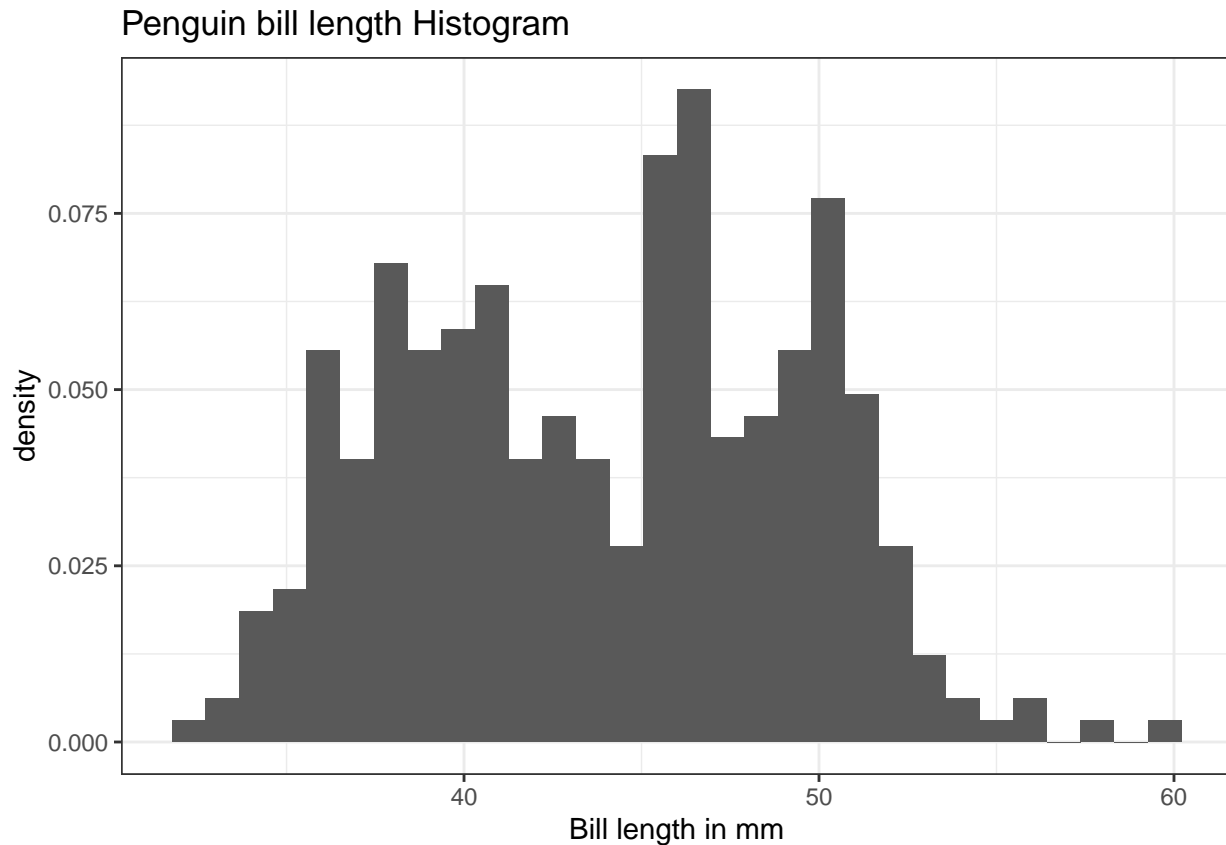


Now we have a histogram like we want it. I will add a title, change the theme and change the axis titles and we can be done for now it's really that simple.

```
ggplot(penguins, aes(x=bill_length_mm, y=..density..)) +  
  geom_histogram() + # this is the line that tells it to make a histogram  
  xlab("Bill length in mm") + # changing the x axis label  
  ggtitle("Penguin bill length Histogram") + # adding the title  
  theme_bw() # changing the theme for fun
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



## 3 The Rep function and sampling

### 3.1 A simple sampling example

Now R is made for statistics and it has amazing random sampling functionality. If you need to sample from something oftentimes you can. Sampling is very important in any statistical practice since it allows us to generate fake data. When I say sampling I essentially mean simulating data. For instance I can simulate a coin toss many times. Let 1 denote heads and 0 denote tails.

```
coin <- c(0,1)
```

From now on for reproducibility I will set a seed (what this does is allows for you to get the exact same random results as I get when you run these functions).

```
set.seed(11)
```

My coin here is simply a vector with a 0 and a 1. I now need a function to randomly pick one of those with equal probability, This is the sample function

```
sample(coin ,size=1)
```

```
## [1] 1
```

So I have drawn heads here, but this is slow for me to run this manually so many times! This is what the size option is for. Let's say I wanted to flip 100 coins.

```
sample(coin, size=100)
```

```
## Error in sample.int(length(x), size, replace, prob): cannot take a sample larger than the population
```

We've received an error that says "cannot take a sample larger than the population when `replace = FALSE`". This is because what R does by default is that it treats your vector like a hat we're pulling papers out of. First it goes in and pulls out a random entry from the vector (in our case this was the 1), but by default it doesn't put this back into the hat, so then the next draw from the vector hat is whatever is left which in our case is just the 0. Finally on the third draw there are no numbers left in the hat to draw from so R spits out an error. We want R to put the papers back into the hat as if we are flipping the coin brand new. We can do this with the `replace=TRUE` argument.

```
flips<-sample(coin, size=100, replace=TRUE)
flips
```

```
## [1] 1 1 0 1 0 0 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 1 0
## [38] 1 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 1 0 0 0 1 0 1
## [75] 1 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0
```

So we've now received our coin draws. Hopefully since we know this is a fair coin (why?), we can assume that we will have about 50 heads. I can just sum to check

```
sum(flips)
```

```
## [1] 51
```

That's pretty close to 50. In addition we can see the proportion of heads by dividing by the length.

```
sum(flips)/length(flips)
```

```
## [1] 0.51
```

An important thing to note is that this is exactly the same as taking the mean which we define as the sum of all our observations divided by the length

```
mean(flips)
```

```
## [1] 0.51
```

## 3.2 More complicated sampling with rep

Suppose I wanted to flip a coin with probability  $1/56$  of landing on heads. We can make this with a vector of 55 zeros and a single one. Note that in defining `weird_coin` I use `rep`, because I'm putting the same value so many times I don't want to write it out so I tell with `rep(0,55)` give me a vector with 55 zeros then with `c(rep(0,55),1)` I append a single one to the new vector.

```
weird_coin <- c(rep(0,55),1)
```

So here I calculate the mean of this (again the same as the proportion as shown above), this should be fairly close to  $1/56$

```
close<-mean(sample(weird_coin,1000,replace=TRUE))
```

```
close-(1/56)
```

```
## [1] -0.005857143
```

It is pretty close.

Someone asked about flipping tails rather than heads well in this case if I made the probability of heads 1, I can just do  $1 - P(\text{heads})$  (the complement rule) or I could replace the vector with the complementary vector. This is just we switch the ones and zeroes.

```
weird_coin_tails <- 1-weird_coin # Why does this work?
```

```
close_tails<-mean(sample(weird_coin_tails,1000,replace=TRUE))
```

```
close_tails-(1-(1/56))
```

```
## [1] 0.003857143
```

Again this is pretty close to what we'd expect it to be.

### 3.3 an extension of what we know (simulating things we don't know the answer to)

Sampling is a rather simple function but we can sometimes make it cater to some more complicated problems however almost all of them can essentially be boiled down to coin flips of some kind. For instance on a previous homework we were mostly doing simulations to confirm something we already knew. Here is an example of simulating with sampling to show something more complicated. All of this can be solved by hand but would be tedious. Don't worry if you don't understand the coding here (in fact its coded rather poorly by me) but this is is trying to see the probability of rolling three die and seeing the probability of getting a dice roll 10 or higher. Inside of the for loop is what is important, I'm sampling three draws with replacement from our dice then summing them together. I am then adding these sums to a vector. After this I take the mean of the number of times the sum was greater than 10. This should approximate the probability of rolling 3 dice and having a sum greater than 10. (this is mathematically supported by something called the law of large numbers).

```
dice<- c(1:6)
```

```
z <- replicate(100000,sum(sample(dice,replace=TRUE,size=3)))
```

```
mean(z>=10)
```

```
## [1] 0.62501
```